HY240: Δομές Δεδομένων
Χειμερινό Εξάμηνο – Ακαδημαϊκό Έτος 2023-2024 Διδάσκουσα: Παναγιώτα Φατούρου
Προγραμματιστική Εργασία – 2η Φάση

Ημερομηνία Παράδοσης: Παρασκευή, 22 Δεκεμβρίου 2023, 23:59
Τρόπος Παράδοσης: Μέσω του προγράμματος turnin. Πληροφορίες για την χρήση του turnin στην ιστοσελίδα τουμαθήματος (https://www.csd.uoc.gr/~hy240/current/submit.php)

Γενική Περιγραφή

Στην εργασία αυτή καλείστε να υλοποιήσετε μία απλοποιημένη υπηρεσία παρακολούθησης ταινιών (streaming service). Η υπηρεσία διαθέτει ταινίες ταξινομημένες σε διαφορετικές θεματικές κατηγορίες. Χρήστες εγγράφονται στην υπηρεσία, παρακολουθούν ταινίες προσθέτοντάς τες στο ιστορικό τους και πραγματοποιούν φιλτραρισμένες αναζητήσεις σε κατηγορίες ταινιών.

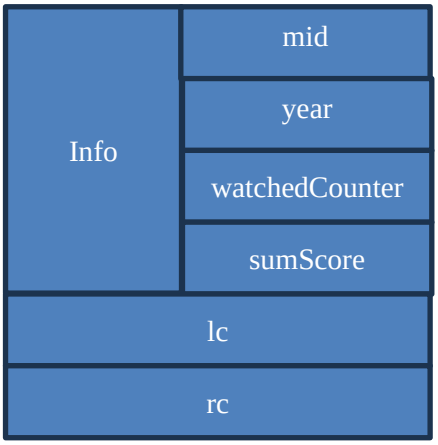
Αναλυτική Περιγραφή Ζητούμενης Υλοποίησης

Η υπηρεσία που θα υλοποιήσετε κατατάσσει τις ταινίες που διαθέτει σε **6** θεματικές κατηγορίες: Horror, Science-Fiction, Drama, Romance, Documentary, Comedy. Κάθε ταινία ανήκει σε **μία μόνο** κατηγορία και διαθέτει ένα **μοναδικό** αναγνωριστικό. Θα υλοποιήσετε την κατηγοριοποίηση των ταινιών μέσω ενός πίνακα 6 θέσεων, που ονομάζεται **πίνακας κατηγοριών**. Σε κάθε θέση του πίνακα βρίσκεται ένας δείκτης (τύπου ***struct movie ****) στη ρίζα ενός δένδρου δυαδικής αναζήτησης (**binary search tree**) που έχει **κόμβο φρουρό**. Το δένδρο μίας κατηγορίας είναι ταξινομημένο ως προς το πεδίο **movieID** των κόμβων του, σύμφωνα με την ενδοδιατεταγμένη διάσχιση (**inorder**), σε **αύξουσα διάταξη**. Κάθε στοιχείο του δένδρου ταινιών μιας συγκεκριμένης κατηγορίας είναι ένα **struct τύπου movie** με τα εξής πεδία:

- **info**: Βοηθητική δομή τύπου ***struct movie_info*** που περιγράφει τις διαθέσιμες πληροφορίες για μία ταινία. Τα πεδία της είναι τα εξής:
    - **mid**: Μοναδικό αναγνωριστικό της ταινίας, τύπου ***int***.
    - **year**: Έτος κυκλοφορίας της ταινίας, τύπου ***int***.
    - **watchedCounter:** αριθμός των χρηστών που έχουν δει την ταινία.
    - **sumScore**: άθροισμα της βαθμολογίας που έχουν δώσει οι χρήστες για την ταινία.
- Ένα δείκτη **lc** που δείχνει στον αριστερό θυγατρικό κόμβο του κόμβου που αντιστοιχείστην ταινία.
- Ένα δείκτη **rc** που δείχνει στον δεξιό θυγατρικό κόμβο του κόμβου που αντιστοιχεί στηνταινία.

Το mid του κόμβου φρουρού έχει αρχική τιμή ίση με -1. Τα πεδία year, watchedCounter και sumScore του κόμβου φρουρού αρχικοποιούνται με την τιμή 0. Οι δείκτες leftChild και rightChild θα αρχικοποιηθούν με την τιμή NULL. Κάθε δένδρο ταινιών κατηγορίας είναι αρχικά άδειο και επομένως περιέχει μόνο τον κόμβο φρουρό.

Στο **Σχήμα 2** φαίνεται ο πίνακας 6 θέσεων με το δένδρο κατηγορίας σε κάθε θέση.

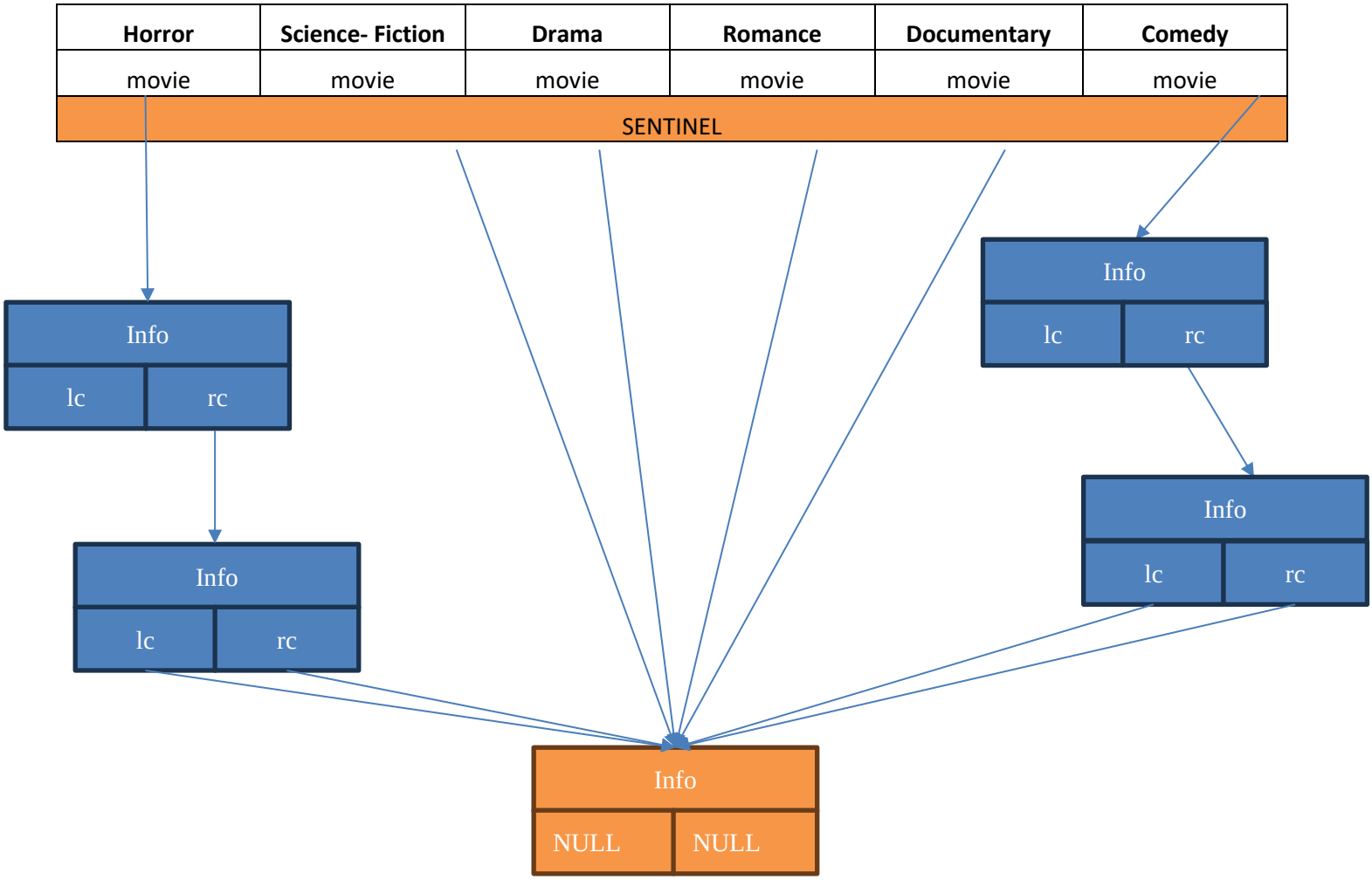


Σχημα 1 Εγγραφή τύπου struct movie

Πριν να εισαχθούν στο κατάλληλο δένδρο ταινιών του πίνακα κατηγοριών, οι καινούριες ταινίες που προστίθενται στην υπηρεσία, εισάγονται σε ένα ξεχωριστό δένδρο, το δένδρο **νέων κυκλοφοριών**. Το δένδρο νέων κυκλοφοριών είναι ένα **δυαδικό δένδρο, ταξινομημένο,** βάση του πεδίου **mid** ,σύμφωνα με την ενδοδιατεταγμένη διάσχιση**. Σε αντίθεση με το δένδρο ταινιών μιας κατηγορίας, το δένδρο νέων κυκλοφοριών δεν έχει κόμβο φρουρό.**

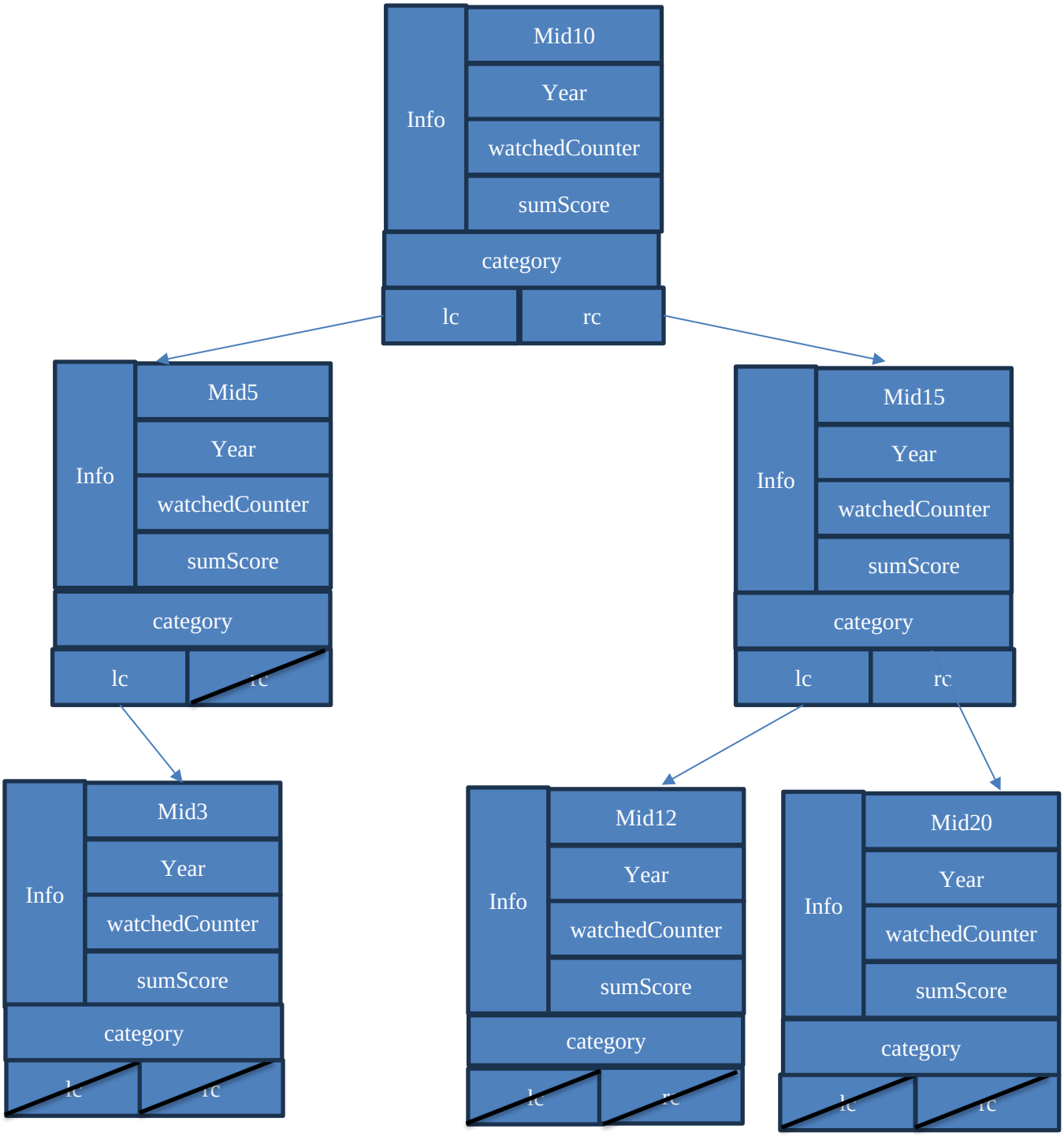
Οι κόμβοι του υλοποιούνται με την δομή ***struct new_movie***, η οποία έχει τα εξής πεδία:

- **info**: Πληροφορίες σχετικά με την ταινία, τύπου ***struct movie_info***, όπως και στο struct movie.
- **category**: Η κατηγορία στην οποία ανήκει αυτή η ταινία, η οποία αναπαρίσταται ως ένα enum τύπου ***movieCategory_t***.
- **lc**: Δείκτης (τύπου ***struct new_movie***), ο οποίος δείχνει στον αριστερό θυγατρικό κόμβο του κόμβου που αντιστοιχεί στην ταινία νέας κυκλοφορίας.
- **rc**: Δείκτης (τύπου ***struct new_movie***), ο οποίος δείχνει στον δεξιό θυγατρικό κόμβο του κόμβου που αντιστοιχεί στην ταινία νέας κυκλοφορίας .



Σχημα 2: : Πίνακας κατηγοριών και δένδρα κατηγοριών

Στο σχήμα 3 βλέπεται πώς μοιάζει το δένδρο νεών κυκλοφοριών σε μια υποθετική εκτέλεση.



Σχημα 3 Δένδρο νέων κυκλοφοριών-ταινιών

Δομές δεδομένων που αφορούν τον χρήστη

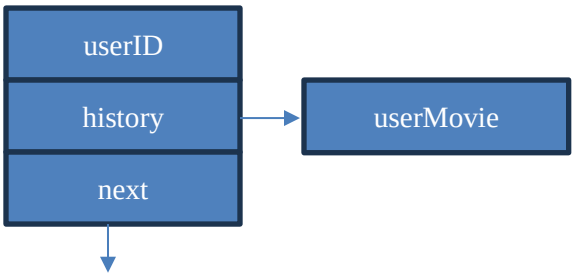
Η υπηρεσία εξυπηρετεί ένα σύνολο εγγεγραμμένων χρηστών. Οι χρήστες θα διατηρούνται σε έναν **πίνακα κατακερματισμού USER[hash_table_size],** ο οποίος περιέχει πληροφορίες για τους χρήστες. Για την επίλυση των συγκρούσεων θα ακολουθήσετε την μέθοδο των **μη-ταξινομημένων αλυσίδων**.

Το μέγεθος του πίνακα κατακερματισμού, **hash_table_size,** θα πρέπει να επιλέγεται από εσάς προσεκτικά και θα πρέπει να είστε σε θέση να δικαιολογήσετε την επιλογή σας.

Κάθε θέση **i, 0 ≤ i < hash_table_size,** περιέχει ένα δείκτη στο πρώτο στοιχείο μιας απλά συνδεδεμένης λίστας που υλοποιεί την αλυσίδα της θέσης i του πίνακα κατακερματισμού. Κάθε στοιχείο μιας αλυσίδας είναι μια εγγραφή (struct), τύπου user, με τα εξής πεδία (δείτε Σχήμα 4):

- Έναν ακέραιο, **userID**, που χαρακτηρίζει μοναδικά το χρήστη.
- Ένα δείκτη, **history**, σε ένα struct τύπου userMovie (βλέπε παρακάτω), που δείχνει σε ένα **διπλά-συνδεδεμένο φύλλο-προσανατολισμένο δένδρο δυαδικής αναζήτησης**, το οποίο ονομάζεται **δένδρο ιστορικού ταινιών του χρήστη**. Το **δένδρο είναι ταξινομημένο** ως προς το πεδίο **movieID σύμφωνα με την ενδοδιατεταγμένη διάσχιση,** και περιέχει ταινίες που έχει ήδη παρακολουθήσει και βαθμολογήσει ο χρήστης.
- Ένα δείκτη, **next**, που δείχνει στο επόμενο στοιχείο της αλυσίδας i .

Προσέξτε ότι το **userID** κάθε χρήστη της αλυσίδας που δεικτοδοτείται από τη θέση i του πίνακα κατακερματισμού, έχει τιμή κατακερματισμού i.



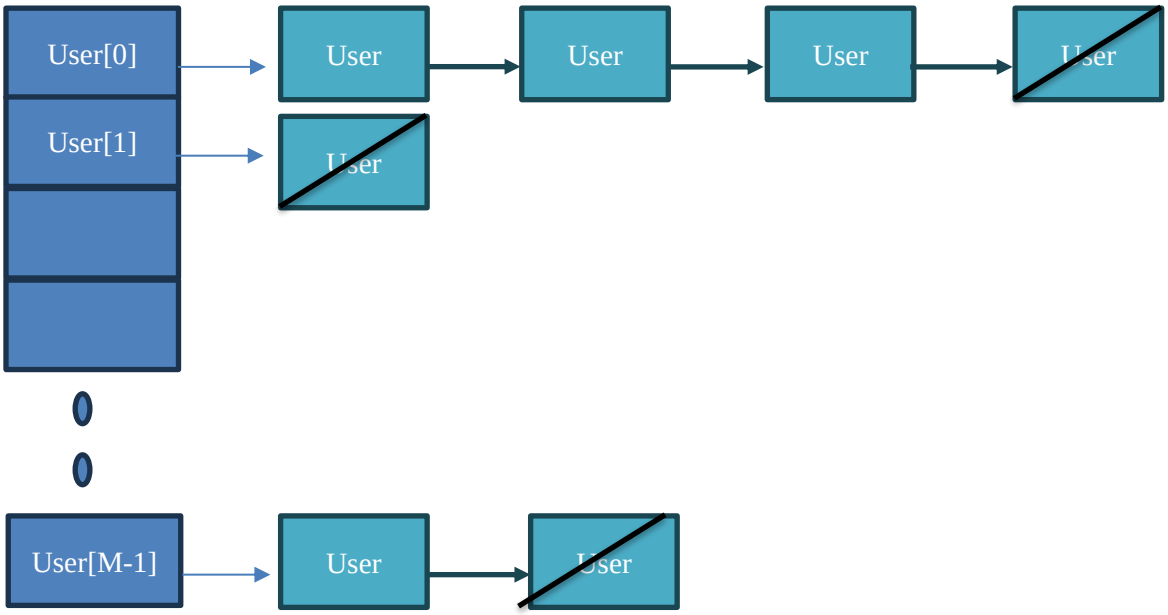
Σχήμα 3: Εγγραφή τύπου User

Για την υλοποίηση της συνάρτησης κατακερματισμού θα πρέπει να βασιστείτε στην τεχνική του **καθολικού κατακερματισμού**. Για την υλοποίηση του καθολικού κατακερματισμού θα δίνονται τα εξής:

1) Ένας πίνακας primes[], ο οποίος περιέχει πρώτους αριθμούς σε αύξουσα σειρά.
2) Το μέγιστο πλήθος χρηστών, μέσω της μεταβλητής max_users
3) Το μέγιστο αναγνωριστικό χρήστη userID, μέσω της μεταβλητής max_id

Αυτές οι μεταβλητές είναι global, έχουν δηλωθεί στο αρχείο Movie.h **και θα αρχικοποιούνται στη main βάσει τιμών που αναγράφονται στις πρώτες γραμμές κάθε test_file.**

Ο πίνακας κατακερματισμού χρηστών παρουσιάζεται στο Σχήμα 4.

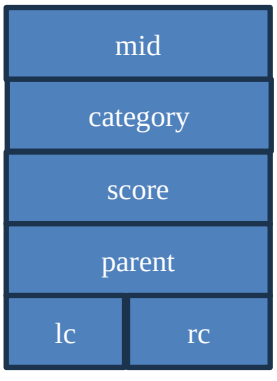


Σχημα 4 Πίνακας κατακερματισμού χρηστών χρησιμοποιώντας την μέθοδο

Κάθε κόμβος του δένδρου ιστορικού ταινιών ενός χρήστη αντιστοιχεί σε μια εγγραφή τύπου userMovie. Το **struct userMovie** περιέχει τα εξής πεδία:

- Έναν ακέραιο **mid** που χαρακτηρίζει μοναδικά την ταινία
- Έναν ακέραιο **category** που αντιστοιχεί στη θεματική κατηγορία της ταινίας. Η μεταβλητή αυτή λαμβάνει τιμές από 0 έως 5, όπου 0: Horror, 1: Science- Fiction, 2: Drama, 3:Romance, 4:Documentary, 5:Comedy.
- Έναν ακέραιο **score** που αντιπροσωπεύει την βαθμολογία που έδωσε ο χρήστης στη ταινία. Η μεταβλητή αυτή παίρνει τιμές στο διάστημα 1 έως 10, με 1 να είναι η χαμηλότερη βαθμολογία για μια ταινία και 10 η μεγαλύτερη.
- Έναν δείκτη **parent** που δείχνει στον πατέρα του κόμβου
- Έναν δείκτη **lc** που δείχνει στον αριστερό θυγατρικό κόμβο.
- Έναν δείκτη **rc** που δείχνει στον δεξιό θυγατρικό κόμβο.

Η εγγραφή τύπου userMovie παρουσιάζεται στο Σχήμα 5.



Σχημα 5: Εγγραφή τύπου userMovie

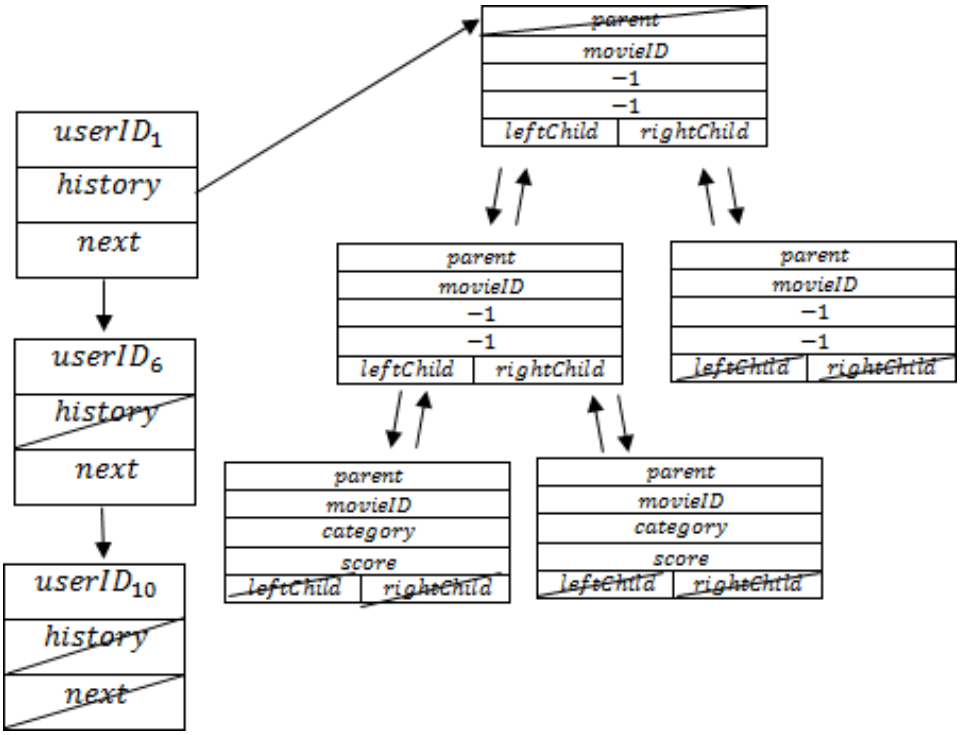
Το *δένδρο ιστορικού* είναι **διπλά-συνδεδεμένο φυλλο-προσανατολισμένο δένδρο δυαδικής αναζήτησης**. Τα φυλλοπροσανατολισμένα δένδρα δυαδικής αναζήτησης (**leaf-oriented binary search trees)** αποτελούν μια εναλλακτική υλοποίηση του αφηρημένου τύπου δεδομένων του λεξικού. Ορίζονται ως εξής:

a) Όλα τα κλειδιά του λεξικού αποθηκεύονται στα φύλλα του δένδρου, από αριστερά προς τα δεξιά κατά μη φθίνουσα τιμή κλειδιού.

b) Οι εσωτερικοί κόμβοι αποθηκεύουν κλειδιά (που δεν αντιστοιχούν απαραίτητα σε κλειδιά του λεξικού), έτσι ώστε να ισχύει η κάτωθι αμετάβλητη συνθήκη σε κάθε κόμβο v:

Το κλειδί του αριστερού παιδιού του v είναι μικρότερο από αυτό του v, ενώ το δεξιό παιδί του v διαθέτει κλειδί μεγαλύτερο ή ίσο από εκείνο του v.

Παρατηρήστε ότι βάσει του ορισμού, οι εσωτερικοί κόμβοι έχουν και τους δύο δείκτες μη κενούς, ενώ και οι δύο δείκτες των φύλλων είναι κενοί. Επομένως ένα φυλλοπροσανατολισμένο δέντρο είναι γεμάτο. Το δένδρο ιστορικού ενός χρήστη που δεικτοδοτείται από κάποιον από τους κόμβους μιας αλυσίδας του πίνακα κατακερματισμού παρουσιάζεται στο Σχήμα 6.

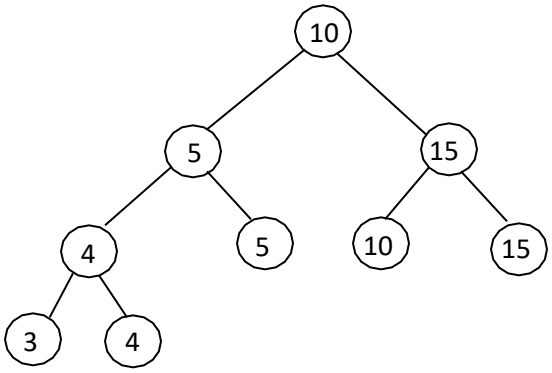


Σχημα 6: Δένδρο ιστορικού ενός χρήστη που δεικτοδοτείται από κάποιον από τους κόμβους μιας αλυσίδας του πίνακα κατακερματισμού

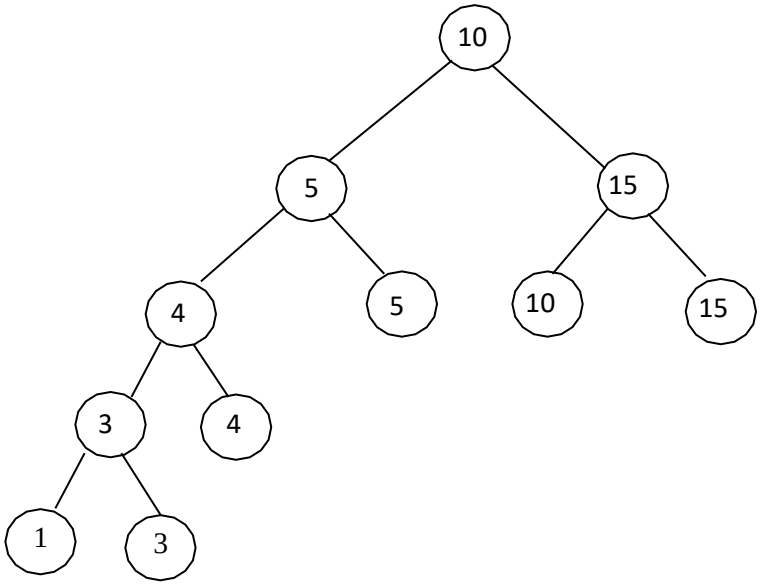
Κανόνες εισαγωγής και διεγράφης σε φυλλοπροσανατολισμένο δέντρο.

1. Για να **εισάγουμε** έναν νέο κόμβο, **v**, με κλειδί K σε ένα **φυλλοπροσανατολισμένο δένδρο** δυαδικής αναζήτησης, εκτελούμε αναζήτηση για να βρούμε το φύλλο, v', το οποίο θα έπρεπε να αποτελέσει τον γονικό κόμβο του v στο δένδρο. Ωστόσο, το κλειδί, K', του v' πρέπει να εξακολουθήσει να εμφανίζεται σε φύλλο του δένδρου. Για να επιτευχθεί αυτό, αντικαθιστούμε τον v' από ένα **δένδρο τριών κόμβων**, το οποίο αποτελείται από έναν εσωτερικό κόμβο με δύο παιδιά φύλλα. Το αριστερό από τα δύο αυτά φύλλα έχει κλειδί min{K, K'}, ενώ το δεξιό φύλλο καθώς και ο v έχουν κλειδί max {K, K'} (**Σχήμα 6**).
2. Για να **διαγράψω** έναν κόμβο, v, από ένα φυλλοπροσανατολισμένο δένδρο δυαδικής αναζήτησης, βρίσκω τον γονικό του κόμβο, v', καθώς επίσης και τον γονικό κόμβο του v', έστω v''. Για τη διαγραφή, αντικαθιστώ τον δείκτη του v'' που δείχνει στον v', ώστε αυτός να δείχνει στον αδελφικό κόμβο του v.

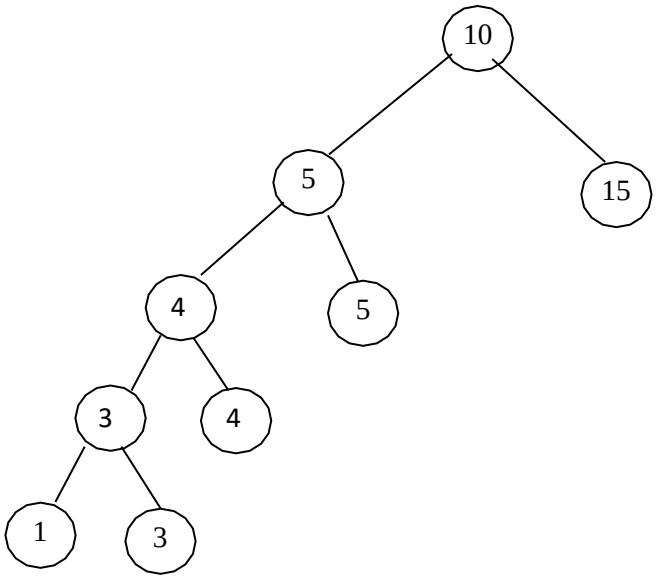
Ένα παράδειγμα φύλλο-προσανατολισμένου δένδρου παρουσιάζεται στο Σχήμα 7, ενώ παραδείγματα εισαγωγής και διαγραφής σε φυλλοπροσανατολισμένο δένδρο παρουσιάζονται στα Σχήματα 8 και 9.



Σχημα 7: Παράδειγμα ενός φύλλο-προσανατολισμένο δένδρο δυαδικής αναζήτησης



Σχημα 8 : Εισαγωγή κλειδιού 1 στο δένδρο του σχήματος 6



Σχημα 9: Διεγράφη κλειδιού 10

Τρόπος Λειτουργίας Προγράμματος

Το πρόγραμμα που θα δημιουργηθεί θα πρέπει να εκτελείται καλώντας την ακόλουθη εντολή:

<executable> <input-file>

όπου <executable> είναι το όνομα του εκτελέσιμου αρχείου του προγράμματος (π.χ. a.out) και <input-file> είναι το όνομα ενός αρχείου εισόδου (π.χ. testfile) το οποίο περιέχει τα γεγονότα.

Τα γεγονότα εισόδου είναι τα εξής:

R <userID >

Γεγονός τύπου ***register user*** το οποίο σηματοδοτεί την εγγραφή ενός νέου χρήστη (user) με αναγνωριστικό <userID>. Το γεγονός αυτό προσθέτει το νέο χρήστη στον πίνακα κατακερματισμού χρηστών της υπηρεσίας. Το πεδίο history του χρήστη πρέπει να έχει την αρχική τιμή NULL.

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

```
R <userID>
Chain <j> of Users:
    <userID1>
    <userID2>
       ...
    <userIDn>
DONE
```

όπου <j> είναι η τιμή κατακερματισμού του κλειδιού <userID>, n είναι ο αριθμός των χρηστών στην αλυσίδα που δεικτοδοτείται από τη θέση <j> του πίνακα Users και για κάθε i ∈ {1, …, n}, <userIDi> είναι το αναγνωριστικό του χρήστη που αντιστοιχεί στον i-οστό κόμβο της αλυσίδας αυτής.

U < userID >

Γεγονός τύπου ***unregister user*** το οποίο σηματοδοτεί τη διαγραφή ενός χρήστη (user) με αναγνωριστικό <userID> από τον πίνακα κατακερματισμού χρηστών. Πριν την οριστική διαγραφή του χρήστη από τη λίστα χρηστών θα πρέπει να διαγράψετε όλα τα στοιχεία του δένδρου ιστορικού ταινιών του χρήστη, αν αυτό περιέχει στοιχεία.

Κατά το γεγονός αυτό εντοπίζεται η κατάλληλη αλυσίδα βάσει της συνάρτησης κατακερματισμού και στη συνέχεια εκτελείται αναζήτηση ώστε να βρεθεί ο κατάλληλος κόμβος της αλυσίδας.

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

```
U <userID>
Chain <j> of Users:
    <userID1>
    <userID2>
       ...
    <userIDn>
DONE
```

όπου <j> είναι η τιμή κατακερματισμού του κλειδιού <userID>, n είναι ο αριθμός των χρηστών στην αλυσίδα που δεικτοδοτείται από τη θέση <j> του πίνακα Users και για κάθε i ∈ {1, …, n}, <userIDi> είναι το αναγνωριστικό του χρήστη που αντιστοιχεί στον i-οστό κόμβο της αλυσίδας αυτής.

A <mid > <category> <year>

Γεγονός τύπου ***add new movie****,* το οποίο σηματοδοτεί την άφιξη μιας νέας ταινίας που είναι διαθέσιμη στους χρήστες. Κατά το γεγονός αυτό, θα δημιουργείται μια νέα ταινία με αναγνωριστικό <movieID> και έτος κυκλοφορίας <year>, η οποία θα ανήκει στη θεματική κατηγορία <category>. **Ανεξάρτητα από την κατηγορία στην οποία ανήκει, η νέα ταινία θα εισάγεται στο δένδρο νέων κυκλοφοριών.** Τα πεδία watchedCounter και sumScore θα αρχικοποιούνται με την τιμή 0.

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος, το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

```
A <movieID> <category> <year>
New releases Tree:
   <new_releases>: <movieID1>, ... , < movieIDn>
DONE
```

όπου n είναι ο αιρθμός των κόμβων του δένδρου νέων κυκλοφοριών. Οι κόμβοι θα πρέπει να έχουν εισαχθεί με τέτοιο τρόπο ώστε αν πραγματοποιηθεί **ενδοδιατεταγμένη** (in-order) **διάσχιση στο δένδρο**, οι ταινίες να προσπελάζονται σε **αύξουσα διάταξη ως προς το πεδίο movieID.**

D

Γεγονός τύπου ***Distribute movies*** το οποίο σηματοδοτεί την κατηγοριοποίηση των ταινιών που περιέχει το δένδρο νέων κυκλοφοριών στις υπόλοιπες θεματικές κατηγορίες. Σε αυτό το γεγονός, θα διασχίζετε το δένδρο των νέων κυκλοφοριών και για κάθε κόμβο, v, που βρίσκετε σε αυτό **θα εισάγετε** έναν νέο κόμβο στο δένδρο της κατάλληλης θεματικής κατηγορίας. Στη συνέχεια, **θα διαγράφετε** τον v από το δένδρο νέων κυκλοφοριών. Προσοχή το δένδρο της εκάστοτε κατηγορίας θα πρέπει να έχει ύψος O(logn) (δείτε σειρά ασκήσεων 3 για να δημιουργήσετε δένδρα με το

κατάλληλο ύψος).

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος, το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

```
D
Movie Category Array:
    <category0>: <movieID01>, ... , < movieID0n0>
    <category1>: <movieID11>, ... , < movieID1n1>
    ...
   <category5>: <movieID51>, ... , < movieID5n4>
DONE
```

όπου για κάθε i, 0 ≤ i ≤ 5, n_i είναι το μέγεθος του δένδρου ταινιών της κατηγορίας i του πίνακα κατηγοριών και για κάθε j, 1 ≤ j ≤ n_i, $<movieID^i_j>$ είναι το αναγνωριστικό της ταινίας που αντιστοιχεί στον j-οστό κόμβο του δένδρου ταινιών της κατηγορίας i, όπως προκύπτει από την ενδοδιατεταγμένη διάσχισή του δένδρου αυτού.

I <movieID> <category>

Γεγονός τύπου *search movie* το οποίο σηματοδοτεί την αναζήτηση της ταινίας με αναγνωριστικό <movieID> στο δένδρο ταινιών της κατηγορίας <category>.

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος, το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

```
I <movieID> <category> <year>
DONE
```

όπου <year> είναι το έτος κυκλοφορίας της ταινίας με αναγνωριστικό <movieID>.

W <userID > <category><movieID> <score>

Γεγονός τύπου *watch movie* το οποίο σηματοδοτεί ότι ο χρήστης με αναγνωριστικό <userID> έχει παρακολουθήσει την ταινία με αναγνωριστικό <movieID> και την αξιολογεί με βαθμό <score>.

Κατά το γεγονός αυτό, θα γίνεται αναζήτηση της ταινίας με αναγνωριστικό <movieID> στο δένδρο κατηγοριών της κατηγορίας <category>. Όταν βρεθεί ο κόμβος της ταινίας, το πεδίο watchedCounter θα αυξάνεται κατά ένα και το πεδίο sumScore θα αυξάνεται κατά την τιμή <score>.

Στη συνέχεια, θα δημιουργείτε έναν κόμβο, userMovie. Ο κόμβος αυτός θα έχει στα πεδία movieID και category ίδιες τιμές με εκείνες του struct που αντιστοιχεί στην ταινία με αναγνωριστικό movieID.

Το πεδίο score του κόμβου userMovie θα έχει την τιμή <score>. Στη συνέχεια θα εισάγετε αυτόν τον κόμβο, στο φυλλο-προσανατολισμένο δένδρο ιστορικού του χρήστη με αναγνωριστικό <userID>.

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος, το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

```
W<userID><category><movieID><score>
History Tree of user <userID>:
      <userMovieID1, score1>
      <userMovieID2, score2>
      …
      <userMovieIDn, scoren>
DONE
```

όπου n είναι το μέγεθος του δένδρου ιστορικού ταινιών του χρήστη με αναγνωριστικό <userID>, για κάθε i ∈ {1, …, n}, <movieIDi> είναι το αναγνωριστικό της ταινίας που αντιστοιχεί στον i-οστό κόμβο του δένδρου, όπως προκύπτει από την ενδοδιατεταγμένη διάσχισή του και <scorei> είναι η βαθμολογία της ταινίας που αντιστοιχεί στον κόμβο αυτό.

F <uid > <score>

Γεγονός τύπου *filter movies,* στο οποίο ο χρήστης ζητάει από την υπηρεσία να του προτείνει ταινίες που ανήκουν σε οποιαδήποτε κατηγορία και να έχουν βαθμολογία μεγαλύτερη ή ίση του <score>. Θα πρέπει να διατρέξετε τα δένδρα των κατηγοριών και να βρείτε τον αριθμό των ταινιών, numMovies, που έχουν μέση βαθμολογία μεγαλύτερη του <score>. Η μέση βαθμολογία μιας ταινίας υπολογίζεται ως εξής:

$$\mu\acute{\epsilon}\sigma\eta_{\beta\alpha\theta\mu o\lambda o\gamma\acute{\iota}\alpha} = \frac{sumScored}{watchedCounter}$$

Στην συνέχεια, θα πρέπει να χρησιμοποιήσετε ένα βοηθητικό πίνακα μεγέθους ίσο με τον αριθμό numMovies. Έπειτα θα κάνετε διάσχιση του δένδρου ταινιών της κάθε κατηγορίας <category> και θα αποθηκεύετε δείκτες προς τους κόμβους του δένδρου που αντιστοιχούν σε ταινίες με μέση βαθμολογία μεγαλύτερη του score, στο βοηθητικό πίνακα. Τέλος, θα εφαρμόζετε τον αλγόριθμο heapsort, για να ταξινομήσετε τον πίνακα βάσει της μέσης βαθμολογίας των ταινιών που αποθηκεύει.

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος, το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

```
F <uid><score>
      <movieID0> <avgScore0> … <movieIDn-1> <avgScoren-1>
DONE
```

όπου για κάθε i, $0 \leq i \leq n-1$, k_i j, $1 \leq j \leq n_i$, <movieID$_i$> είναι το αναγνωριστικό της ταινίας που αντιστοιχεί στη i- οστή θέση του βοηθητικού πίνακα μετά την εφαρμογή του αλγόριθμου heapsort.

Q <userID>

Γεγονός τύπου *users' average rate,* το οποίο σηματοδοτεί τον υπολογισμό και την εκτύπωση στατιστικών για τη βαθμολογία του χρήστη με αναγνωριστικό <userID>. Πιο συγκεκριμένα, σε αυτό το γεγονός θα πρέπει να ακολουθήσετε τα εξής βήματα:

Εντοπίζετε την κατάλληλη αλυσίδα χρήστη, βάσει της συνάρτησης κατακερματισμού και εκτελείτε αναζήτηση ώστε να βρεθεί ο κόμβος της αλυσίδας που αντιστοιχεί στον χρήστη με αναγνωριστικό <userID>.

Στη συνέχεια, διασχίζετε το δένδρο ιστορικού αυτού του χρήστη και αποθηκεύετε σε μια βοηθητική μεταβλητή, ScoreSum, το άθροισμα των πεδίων score των κόμβων του δένδρου. Επιπρόσθετα, αποθηκεύετε σε μια άλλη βοηθητική μεταβλητή, counter, το πλήθος των ταινιών που είναι αποθηκευμένες στο δένδρο. Τέλος, διαιρείτε το ScoreSum με τον counter για να βρείτε τη μέση βαθμολογία που έχει δώσει ο χρήστης με αναγνωριστικό <userID> στις ταινίες που έχει παρακολουθήσει.

Η διάσχιση των φύλλων του φυλλο-προσανατολισμένου δένδρου θα πρέπει να πραγματοποιείται ως εξής: Αρχικά, θα βρίσκετε το αριστερότερο φύλλο, v, στο δένδρο. Για να βρείτε το φύλλο με το αμέσως μεγαλύτερο κλειδί από εκείνο του v, θα πρέπει να υλοποιήσετε αλγόριθμο, FindNextLeaf(), ο οποίος θα παίρνει ως παράμετρο έναν δείκτη στον κόμβο v και θα εκτελείται σε χρόνο O(h). Η διάσχιση των φύλλων θα γίνεται καλώντας επαναληπτικά την FindNextLeaf() ξεκινώντας από το αριστερότερο φύλλο και μέχρι εξαντλήσεως των φύλλων.

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος, το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

```
Q <userID><MScore>
DONE
```

όπου MScore, είναι ο μέσος όρος βαθμολογίας στις ταινίες που έχει παρακολουθήσει ο χρήστης με αναγνωριστικό <userID> , όπως υπολογίζεται παραπάνω.

M

Γεγονός τύπου *print movies* το οποίο σηματοδοτεί την εκτύπωση του δένδρου ταινιών όλων των κατηγοριών. Σε αυτό το γεγονός θα πρέπει για κάθε κατηγορία να εκτελέσετε ενδοδιατεταγμένη διάσχιση στο δένδρο και να τυπώσετε τους κόμβους που διασχίζετε.

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

```
M
Movie Category Array:
    <category0>: <movieID01>, ... , < movieID0n0>
    <category1>: <movieID11>, ... , < movieID1n1>
    ...
   <category5>: <movieID51>, ... , < movieID5n4>
DONE
```

όπου για κάθε i, $0 \le i \le 4$, n_i είναι το μέγεθος του δένδρου ταινιών της κατηγορίας i του πίνακα κατηγοριών και για κάθε j, $1 \le j \le n_i$, <movieID^{i}j> είναι το αναγνωριστικό της ταινίας που αντιστοιχεί στον j-οστό κόμβο του δένδρου ταινιών της κατηγορίας i, όπως προκύπτει από την ενδοδιατεταγμένη διάσχισή του δένδρου αυτού.

P

Γεγονός τύπου *print users* το οποίο σηματοδοτεί την εκτύπωση του πίνακα κατακερματισμού χρηστών. Για τον κάθε χρήστη θα πρέπει να εκτυπώνονται όλα τα πεδία του struct που του αντιστοιχεί (εκτός από τους δείκτες), συμπεριλαμβανομένου του δένδρου ιστορικού.

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος, το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

```
P

…

Chain <j> of Users:
       <userID_1>
        History Tree:
              <movieID^1_1> <score^1_1>
              …
              <movieID^1_n1> <score^1_n1>
       <userID_2>
        History Tree:
              <movieID^2_1> <score^2_1>
              …
              <movieID^2_n2> <score^2_n2>
       …
       <userID_n>
        History Tree:
              <movieID^n_1> <score^n_1>
              …
              <movieID^n_nm> <score^n_nm>
…
DONE
```

όπου <j> είναι η j-στη αλυσίδα του πίνακα κατακερματισμού και n είναι ο αριθμός των χρηστών στην αλυσίδα που δεικτοδοτείται από τη θέση <j> του πίνακα Users και για κάθε i ∈{1, …, n}, <userIDi> είναι το αναγνωριστικό του χρήστη που αντιστοιχεί στον i-οστό κόμβο της αλυσίδας αυτής. Επίσης, , 1 ≤ k ≤ ni, <movieIDik> είναι η ταίνια από το δένδρο ιστορικού του χρήστη με αναγνωριστικό <userIDi>. Σε αυτό το γεγονός θα πρέπει να τυπώνεται ολόκληρος ο πίνακας κατακερματισμού.

Δομές Δεδομένων

Στην υλοποίησή σας δεν επιτρέπεται να χρησιμοποιήσετε έτοιμες δομές δεδομένων (πχ., ArrayList) είτε η υλοποίηση πραγματοποιηθεί στη C, C++ είτε στη Java. Στη συνέχεια παρουσιάζονται οι δομές σε C που πρέπει να χρησιμοποιηθούν για την υλοποίηση της παρούσας εργασίας.

```
/**
* Structure defining a node of movie binary tree (dendro tainiwn kathgorias)
*/
typedef struct movie{
    int movieID;                /* The movie identifier*/
    int category;               /* The category of the movie*/
    int year;                   /* The year movie released*/
    int watchedCounter;         /* How many users rate the movie*/
    int sumScore;               /* The sum of the ratings of the movie*/
    struct movie *lc;           /* Pointer to the node's left child*/
    struct movie *rc;           /* Pointer to the node's right child*/
}movie_t;


/**
* Structure defining movie_category
*/

typedef struct movie_category{
    movie_t *movie;             /* Pointer to movie tree */
    movie_t *sentinel;          /* Pointer to movie tree sentinel */
}movieCategory_t;

/**
* Structure defining a node of user_movie for history doubly linked binary
* tree (dentro istorikou)
*/
typedef struct user_movie{
    int movieID;                    /* The movie identifier*/
    int category;                   /* The category of the movie*/
    int score;                      /* The score of the movie*/
    struct user_movie *parent;      /* Pointer to the node's parent*/
    struct user_movie *lc;          /* Pointer to the node's left child*/
    struct user_movie *rc;  /* Pointer to the node's right child*/
}userMovie_t;

/**
* Structure defining a node of users' hashtable (pinakas katakermatismou
* xrhstwn)
*/
typedef struct user {
    int userID;             /* The user's identifier*/
    userMovie_t *history;   /* A doubly linked binary tree with the movies
watched by the user*/
    struct user *next;      /* Pointer to the next node of the chain*/
}user_t;

/* Global variables for simplicity. */
```

```
movieCategory_t *categoryArray[6];  /* The categories array (pinakas
kathgoriwn)*/
movie_t *new_releases;     /* New releases simply-linked binary tree*/
user_t **user_hashtable_p; /* The users hashtable. This is an array of
chains (pinakas katakermatismoy xrhstwn)*/
int hashtable_size;         /* The size of the users hashtable)*/
int max_users;              /* The maximum number of registrations (users)*/
int max_id;                 /* The maximum acount ID */
int primes_g[160];          /* Prime numbers for hashing*/
```